Problèmes de satisfaction de contraintes

Systèmes décisionnels et programmation avancée III

Philippe Muller

2012-2013

Plan du cours

- satisfaction de contraintes et applications
- définitions
- la consistance d'arcs
- algorithmes classiques : back-track, forward-checking, MAC
- méthodes par ordonnancement de variables
- méta-heuristiques et contraintes

Systèmes de contraintes

- version encore plus simplifiée formellement d'un processus de raisonnement
- uniquement des "faits" : variables avec domaines restreints + relations entre variables
- recherche d'instances des variables satisfaisant les relations ("contraintes")
- applications : emploi du temps, occupation de salles, gestion de configuration
- un cas particulier de recherche dans un espace de solutions combinatoires (cf espace d'états)
- correspond à des cas où les formes des solutions sont difficiles à prévoir
- existence d'algorithmes spécialisés

Un exemple simple : le “coloriage” de graphe

G $= (N, A)$ un graphe donné par des sommets $u \in N$ et des arcs $(u, v) \in A$
Un ensemble de couleurs C, de taille $|C| = p$
Coloriage = attribuer un élément de C à chaque sommet, $c(u)$, tel que

$$(u, v) \in A \rightarrow c(u) \neq c(v)$$

s’applique à tous les problèmes de gestion d’incompatibilité : par exemple assigner des employés à des bureaux, sans mettre ensemble des gens qui ne s’entendent pas.

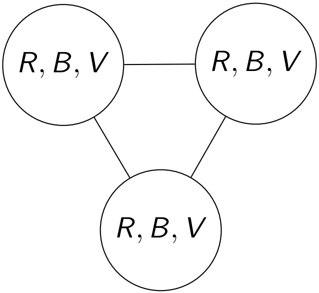
Systèmes de contraintes

- des variables $\{V_1, V_2, V_3, ...\}$
- des domaines de valeurs : $V_1 \in D_1, V_2 \in D_2, ...$
- des contraintes entre variables : ex $V_1 \neq V_2,\ V_3 \leq V_4$
- qui définissent des tuples possibles de valeurs sur les domaines : $C_{ij...} = \{(x_i, y_j, ...) \in D_i \times D_j \times ...\}$

Une modélisation du coloriage

$N = \{u_1, ..., u_n\}$
$C = \{c_1, ..., c_p\}$
posons $|A| = m$
variables V_i = couleur du sommet u_i
pour tout i, $V_i \in C$. les domaines sont tous les mêmes $D_i = C$
contraintes : pour chaque arc (u_i, u_j) de A, $V_i \neq V_j$
(m contraintes, entre 2 variables seulement pour chaque)

Exemple



Représentation explicite des contraintes

- contrainte sur variables = liste d'assignations possibles de ces variables
- exemple sur le coloriage, contrainte sur u_1 et u_2 : $C_{1,2} = \{(R, B), (R, V), (B, V), (B, R), (V, B), (V, R)\}$

Restrictions

- domaines des variables finis
- contraintes **binaires**, $C_{x_i x_j}$ entre 2 variables x_i et x_j

NB : si les domaines sont finis, toute contrainte entre n variables peut se réexprimer avec des contraintes binaires (pas forcément simplement).

Caractéristiques de la formulation par contraintes

- très générale : permet d'encoder de nombreux problèmes
- très utilisée en pratique
- mais : inclut des problèmes difficiles (*NP-difficiles*)

Deux aspects

- modélisation
  - choisir les bonnes variables !
  - influe sur le nombre de contraintes
  - influe sur la complexité des contraintes
  - $\rightarrow$ influe sur la difficulté à résoudre
- résolution : un compromis entre
  - l'inférence sur les domaines des variables (*propagation de contraintes*)
  - l'exploration des possibilités d'affectation

Propagation de contraintes : consistance d'arc

"arc consistency"
On peut éliminer les valeurs d'un domaine qui ne sont dans aucune solution
Exemple : si $D_1 = \{1, 2, 3, 4\}$ et $D_2 = \{1, 2, 3, 4\}$
et que la contrainte est $X_1 > X_2$
que peut-on éliminer ?

"arc consistency"

On peut éliminer les valeurs d'un domaine qui ne sont dans aucune solution

Exemple : si $D_1 = \{1, 2, 3, 4\}$ et $D_2 = \{1, 2, 3, 4\}$
et que la contrainte est $X_1 > X_2$
que peut-on éliminer ?
de D_1 : 1

Propagation de contraintes : consistance d'arc

"arc consistency"

On peut éliminer les valeurs d'un domaine qui ne sont dans aucune solution

Exemple : si $D_1 = \{1, 2, 3, 4\}$ et $D_2 = \{1, 2, 3, 4\}$

et que la contrainte est $X_1 > X_2$

que peut-on éliminer ?

de D_1 : 1

de D_2 : 4

Propagation de contraintes : consistance d'arc

Définition

une contrainte/un arc $\langle X, Y \rangle$entre deux variables X, Y de domaines D_X, D_Y est arc-consistante si et seulement si

- pour toute valeur de D_X, il existe une valeur de D_Y qui satisfait la contrainte
- pour toute valeur de D_Y, il existe une valeur de D_X qui satisfait la contrainte

Propagation de contraintes : consistance d'arc

Définition

une contrainte/un arc $\langle X, Y \rangle$entre deux variables X, Y de domaines D_X, D_Y est arc-consistante si et seulement si

- pour toute valeur de D_X, il existe une valeur de D_Y qui satisfait la contrainte
- pour toute valeur de D_Y, il existe une valeur de D_X qui satisfait la contrainte

Exemple : si $D_1 = \{1, 2, 3, 4\}$ et $D_2 = \{1, 2, 3, 4\}$
et que la contrainte est $X_1 > X_2$
pour 1$\in D_1$, aucun élément de D_2 ne satisfait la contrainte
pour 4$\in D_1$, aucun élément de D_2 ne satisfait la contrainte

Propagation de contraintes : consistance d'arc

$\rightarrow$ un moyen simple de simplifier la résolution : forcer l'arc-consistance, en éliminant les valeurs qui ne la respectent pas, pour chaque arc
on garde une table dynamique de domaines
(initialisée par copie des domaines complets)

```
procedure REVISE1(X,Y)
    for all x_i ∈ D_X do
        if il n'y a pas de y_i ∈ Y tel que ⟨x_i, y_i⟩ ∈ C_ij then
            D_X = D_X / {x_i}
        end if
    end for
    if D_X a été modifié then
        renvoie Vrai
    end if
end procedure
```

```
procedure REVISE2(X,Y)
    for all  y_i ∈ D_Y  do
        if  il n'y a pas de x_i ∈ X tel que ⟨x_i, y_i⟩ ∈ C_ij then
            D_Y = D_Y/{y_i}
        end if
    end for
    if  D_Y a été modifié then
        renvoie Vrai
    end if
end procedure
```

version brutale : AC1 : arc consistency 1

procedure PROPAGE(CSP)
    while un domaine a été purgé **do**
        for all $C_{XY} = (X, Y) \in \text{CSP}$ **do**
            revise1(X, Y)
            revise2(X, Y)
        end for
    end while
end procedure

Propagation de contraintes : consistance d'arc

version plus efficace : AC-3 on garde la trace des domaines modifiés

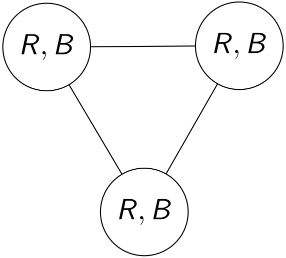
procedure PROPAGE-3(CSP)
    Q = ensemble des variables
    while Q non vide **do**
        change = {}
        for all $C_{XY} = (X, Y) \in$CSP tels que $X \in Q$ *ou* $Y \in Q$ **do**
            if revise1(X, Y) **then**
                $change = change \cup \{X\}$
            end if
            if revise2(X, Y) **then**
                $change = change \cup \{Y\}$
            end if
        end for
        Q = change
    end while
end procedure

Exercices

- dérouler l'algorithme AC3 sur l'instance de coloration suivante : trois sommets A, B, C avec les couleurs possibles, respectivement {r,b,v}, {r,v}, {v}
- modéliser le problèmes des 4-reines sous forme de contraintes
- appliquer la consistance d'arc pour éliminer des affectations dans le problème des 3-reines
- modéliser le problème d'ordonnancement suivant : on a 4 tâches à réaliser de durées 5, 1, 3, 4
  les tâches 2 et 3 ne peuvent être faites en parallèle (mêmes ressources nécessaires), et la 3 doit être faite avant la 4.
  peut-on avoir toutes les tâches commencées avant le temps t=5 ?

Insuffisance de la consistance d'arc

"filtre" des mauvaises solutions, mais ne suffit pas toujours à trouver une solution complète
ne suffit pas non plus à filtrer toutes les mauvaises solutions :



Résolution complète

d'une façon générale, le problème peut être abordé en essayant toutes les assignations possibles (exploration)

$\rightarrow$ problème combinatoire

$\rightarrow$ techniques de résolution générales, cf partie Matthieu Serrurier

en pratique, on va se servir des spécificités du cadre pour raffiner les méthodes générales

notamment en combinant avec la notion de consistance

Méthodes d'exploration générales

espace d'état : des assignations partielles de variables
état initial : CSP complet
état solution : assignation complète cohérente
opérateur sur un état : assigner une valeur à une variable sans violer de contraintes
ordre d'assignation des variables pas importants ? on le fixe pour réduire le nombre d'états
chaque état correspond à une "profondeur" i : nb de variables instanciées

procedure EXPLORE(CSP)
    Q $= \{E_0\}$
    while Q non vide **do**
        courant = choisir_dans(Q)
        if courant n'est pas solution **then**
            new = { instantier la variable $x_{profondeur(courant)}$ }
            Q = Q ∪ new
        else

Quelles options ?

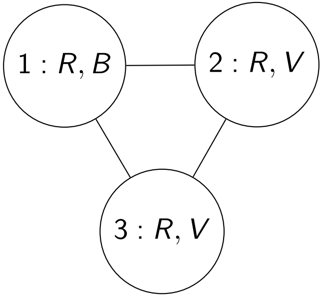
- profondeur/largeur d'abord
- "meilleur" d'abord, selon critère à définir
- ordre des variables en pratique : important pour la performance
- taille de l'espace d'états ?

Profondeur d'abord

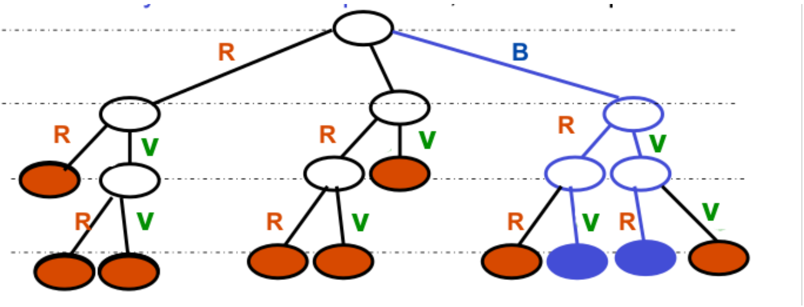
$\equiv$ instancier les valeurs en séquence
une spécificité du problème : on peut arrêter d'explorer si une instantiation est incohérente, et revenir à la valeur précédente ("backtrack") $\rightarrow$ on teste la cohérence à chaque instantiation
version ajustée de l'algorithme
Variables X_i, domaines D_i, ordonnées.

```
procedure BACKTRACK(CSP)
    i = 1                                            ▷ profondeur
    a = {}                 ▷ assignation de valeurs aux variables
    D'_1 = copie(D_1)
    while 1 ≤ i ≤ n do
        x = select_valeur(a,D'_i)     ▷ D'_i est modifié, x étend a
        if x est "rien" then
            i = i - 1                                ▷ backtrack
        else
            i = i + 1
            D'_i = copie(D_i)  ▷ copie le domaine de la var suivante
        end if
    end while
    if i=0 then
        CSP incohérent
    else
        a contient les assignations de variables pour une solution
    end if
end procedure
```

Exemple : coloration



Coloration



Exercice : 3-reines

Exercice : 3-reines

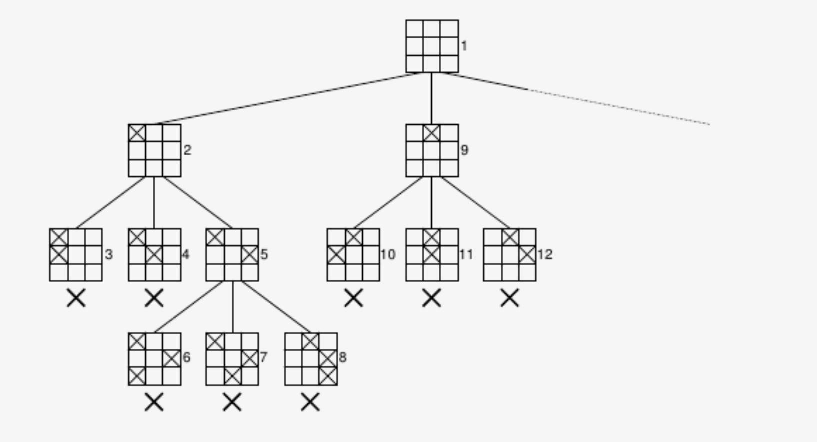


figure : Pierre Lopez

Exercice : 4-reines

Exercice : 4-reines

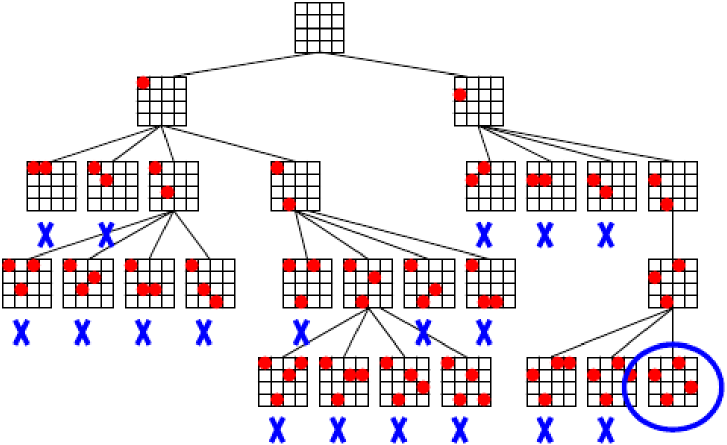


figure : Roman Bartak

Combiner la propagation de contraintes et l'exploration

n = $\|$variables$\|$, c = $\|$contraintes$\|$, k = taille max domaine
à chaque assignation on peut faire une propagation/ test d'arc consistence
mais : question de performances
complexité des procédures ?

ac1 :

ac3 :

d'où compromis : consistance partielle

Combiner la propagation de contraintes et l'exploration

$n = \|\text{variables}\|$, $c = \|\text{contraintes}\|$, $k =$ taille max domaine
à chaque assignation on peut faire une propagation/ test d'arc consistence
mais : question de performances
complexité des procédures ?

ac1 : au pire
(revise * 2 * nb de contraintes) * nb variable * taille max
revise = o(taille max au carré)

ac3 :

d'où compromis : consistance partielle

Combiner la propagation de contraintes et l'exploration

n = ||variables||, c = ||contraintes||, k = taille max domaine
à chaque assignation on peut faire une propagation/ test d'arc consistence
mais : question de performances
complexité des procédures ?

ac1 : au pire
(revise * 2 * nb de contraintes) * nb variable * taille max
revise = o(taille max au carré) nck^3

ac3 :

d'où compromis : consistance partielle

Combiner la propagation de contraintes et l'exploration

n = ||variables||, c = ||contraintes||, k = taille max domaine
à chaque assignation on peut faire une propagation/ test d'arc consistence
mais : question de performances
complexité des procédures ?

ac1 : au pire
(revise * 2 * nb de contraintes) * nb variable * taille max
revise = o(taille max au carré) nck^3

ac3 : on économise de repasser chaque variable

d'où compromis : consistance partielle

Combiner la propagation de contraintes et l'exploration

n = ||variables||, c = ||contraintes||, k = taille max domaine
à chaque assignation on peut faire une propagation/ test d'arc consistence
mais : question de performances
complexité des procédures ?

ac1 : au pire
(revise * 2 * nb de contraintes) * nb variable * taille max
revise = o(taille max au carré) nck^3

ac3 : on économise de repasser chaque variable ck^3

d'où compromis : consistance partielle

Forward-checking

test de consistance "vers l'avant" :

- propager les contraintes concernant la dernière variable instantiée
  (o(ck))
- algo de back-track
- gestion des domaines plus complexes

Application à la coloration

{r,b,v}, {r,v}, {v}

```
procedure FORWARD-CHECKING(CSP)
    i = 0                                          ▷ profondeur
    a = {}                  ▷ assignation de valeurs aux variables
    pour tout j D'_j = copie(D_j)        ▷ copie tous les domaines
    while 1 ≤ i ≤ n do
        x = select_valeur_FC(a,i,{D'_j})   ▷ D'_j modifiables, x étend a
        if x est "rien" then
            remettre tous les domaines filtrés D'_k (k > i)
            à leur valeur d'avant la dernière instantiation de x_i
            i = i - 1                              ▷ backtrack
        else
            i = i + 1
        end if
    end while
    if i=0 then
        CSP incohérent
    else
        a contient les assignations de variables pour une solution
    end if
end procedure
```

procedure SELECT_VALEUR_FC(a,i,{D'$_j$})
    while D'_i non vide **do**
        choisir et enlever $a_j \in D'_i$
        for all k, $i < k \leq n$ **do**
            for all b $\in$ D'$_k$ **do**
                if $a \cup \{x_i : a_j\} \cup \{x_k : b\}$ n'est pas cohérent **then**
                    enlever b de D'$_k$
                end if
            end for
        end for
        if il existe un D'$_k$ vide **then**
            remettre tous les domaines D'$_k$ $(k > i)$
            à leur valeur d'avant le choix de a_j
        else
            retourner a_j
        end if
    end while
    retourner "rien"
end procedure

Exercices

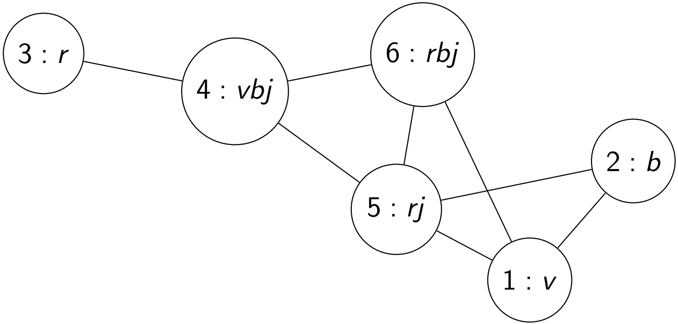
- application n-reines (n=3, n=4)
- faire un pas sur la grille suivante, en instantiant la case (1,3), et en ne considérant que les contraintes liées aux cases du carré en haut à gauche.
- carré magique 3x3 : somme = 15,
  - départ : 4,3,8 // 9
  - départ : 4,3,8 // _ 5

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Ordonnancement de variables

- backtrack fixe arbitrairement l'ordre d'instantiation
- beaucoup de situations où l'on peut faire mieux, en prenant tout de suite les décisions difficiles
  - les variables les plus contraintes (beaucoup de conflits)
  - les variables avec le moins de valeurs disponibles (cf cas des singletons)
- on peut aussi ordonner les valeurs considérées
  - par exemple, prendre les valeurs les moins problématiques (qui éliminent le moins de valeurs des variables reliées)
- pour limiter la malchance, on peut réordonner aléatoirement les variables restantes à chaque retour arrière

Exemple : coloration



variable à prendre d'abord : 5

(+petit domaine, +grd nb de voisins)

valeur à prendre d'abord : rouge

(élimine 1 valeur au lieu de 2 pour jaune)

```
procedure BACKTRACK_AVEC_ORDRE(CSP)
    trier les variables par nombre de contraintes décroissantes où elles
apparaissent,
    puis par taille de domaine croissante en cas d'égalité
    i = 1
    a = {}
    D'1 = copie(D1)
    while 1 ≤ i ≤ n do
        x = select_valeur(a,i,{D'j})          ▷ l'ordre des xi est modifié
        ...
    end while
    ...
end procedure
```

```
procedure SELECT_VALEUR_SIMPLE(a,D'_i)
    while D'_i pas vide do
        choisir et enlever a_j ∈ D'_i
        if a ∪ {x_i : a_j} est cohérent then
            renvoyer a_j
        end if
    end while
    retourner "rien"
end procedure
```

```
procedure SELECT_VALEUR2(a,i,{D'_j})          ▷ ordre sur les variables
    k = min_{j/i<j}(|D'_j|)          ▷ variable restante de plus petit domaine
    échange x_{i+1} et x_k
    while D'_i pas vide do
        choisir et enlever a_j ∈ D'_i
        if a ∪ {x_i : a_j} est cohérent then
            renvoyer a_j
        end if
    end while
    retourner "rien"
end procedure
```

```
procedure SELECT_VALEUR3(a,i,{D'_j})        ▷ ordre sur les valeurs
                                             ▷ cas de la coloration
    while D'_i pas vide do
        v = min_{a_k ∈ D'_i}(∑_{j/i<j,(x_i,xj)∈CSP}(a_k ∈ D'_j))
              ▷ valeur la moins contraignante pour les variables liées à x_i
        enlever v de D'_i
        if a ∪ {x_i : v} est cohérent then
            renvoyer v
        end if
    end while
    retourner "rien"
end procedure
```

Exemples

- coloration
- carré magique : quel est le meilleur choix
  - de première case
  - et de première valeur ?

Exercice : Mariages stables

"Mariage stable" = problème d'appariement avec contraintes

- ex : appariement compagnies/candidats à un poste, ...
- 2 ensembles, et on doit trouver une bijection
- chacun a des préférences partielles (il peut y avoir des ex-aequo), et des vetos
- "stabilité" : que personne ne soit soumis à la tentation de divorcer pour trouver mieux ailleurs, avec quelqu'un qui préfère aussi la personne

- 1 préfère 8, puis (12 et 10 ex aequo) ;
- 2 préfère (8 et 11 ex aequo), puis 12 ;
- 3 préfère 7, puis 9, puis 12 ;
- 4 préfère 12, puis 9 ;
- 5 préfère 8, puis 7, puis 11 ;
- 6 préfère 12, puis (10 et 8 ex aequo), puis 11, puis 7.

Les classements des femmes :

- 7 préfère (5 et 3 ex aequo), puis 6 ;
- 8 préfère 2, puis, 5, puis 1, puis 6 ;
- 9 préfère (3 et 4 ex aequo) ;
- 10 préfère 6, puis 1 ;
- 11 préfère 5, puis 2, puis 6 ;
- 12 préfère 1, puis (4 et 6 ex aequo), puis 2, puis 3.

Questions

modélisation en contraintes :

- variables
- domaines
- contraintes
- ordonner les variables
  - par taille de domaine
  - par nombre de contraintes
- ordonner les valeurs ?
- résoudre par BT + ordonnancement de variables

Backjumping

- une autre source de travail inutile : revenir sur une variable alors que le problème se situe plus haut dans l'arbre de recherche
- amélioration : remonter, en cas de backtrack, à la dernière variable rencontrée qui est en conflit avec la variable courante
- "backjumping"
- "en conflit" = une contrainte impliquant les 2 variables est violée pour certaines valeurs de la variable courante

Méthodes locales appliquées aux contraintes

méthodes “locales” :

- problèmes combinatoires trop gros pour trouver une solution en temps raisonnable
- méthodes approchées en optimisation : trouver quelque chose d”’assez bon”
- méthodes partielles en satisfaction : explorer autour d’une solution partielle
- modifications minimales, graduelles
- compromis temps/qualité
- compromis hasard/exploration

Principe

- générer : une instance, une solution partielle, etc
  pour les contraintes : générer une instanciation de toutes les variables
- évoluer/réparer : changer une partie de l'instance courante, en essayant de l'améliorer :
  pour les contraintes : changer une variable qui viole une contrainte
- nombreux critères possibles, fonction d'évaluation à définir

Stratégie d'évolution : méta-heuristiques

- choix au hasard, modification seulement si il y a une amélioration
- hill-climbing : choix du meilleur changement dans un voisinage donné (à définir)
- tabou : idem avec mécanisme pour éviter de rester dans un optimum local
- recuit simulé, colonies de fourmi, algo génétique etc
- redémarrages aléatoires

Fonctions d'évaluation

- nombre de variables en conflits $\rightarrow$ minimiser les conflits
- nombre de contraintes violées avec un poids par contrainte : à chaque fois qu'on atteint un optimum local, on augmente le poids des contraintes violées $\rightarrow$ minimiser le poids total
- the sky's the limit ...

Schéma d'algorithme

```
A* ← creer_une_solution()
for all  t=1 a max_essais  do
    A ← nouvelle_solution()
    for all  m=1 a max_mouvements do
        A' ← choisir_voisin(A)
        d ← (f(A')-f(A))
        if  acceptable(d) then
            A ← A'
        end if
    end for
    if  f(A*)>f(A) then
        A* ← A
    end if
end for
retourner A*,f(A*)
```

Exemple

- coloration, 3 sommets avec les domaines suivants : rvb, rv, rv
- créer une solution ?
- fonction d'acceptation ?
- choisir un voisin ?

Exemple

- coloration, 3 sommets avec les domaines suivants : rvb, rv, rv
- créer une solution ? instantiation au hasard
- $\rightarrow$ exemple x1=v, x2=r, x3=r
- fonction d'acceptation ?

- choisir un voisin ?

Exemple

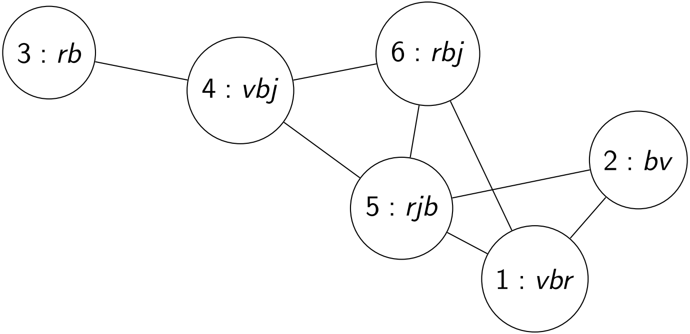
- coloration, 3 sommets avec les domaines suivants : rvb, rv, rv
- créer une solution ? instantiation au hasard
- $\rightarrow$ exemple x1=v, x2=r, x3=r
- fonction d'acceptation ? $|\{(x_i, x_j)/x_i = x_j\}|$
- f = 1
- choisir un voisin ?

Exemple

- coloration, 3 sommets avec les domaines suivants : rvb, rv, rv
- créer une solution ? instantiation au hasard
- $\rightarrow$ exemple x1=v, x2=r, x3=r
- fonction d'acceptation ? $|\{(x_i, x_j)/x_i = x_j\}|$
- f = 1
- choisir un voisin ? "flip" d'une variable x_i, sur une valeur a_k, choisir ce qui minimise la fonction
- flip x1 / r $\rightarrow$ f = 3
  flip x1 / b $\rightarrow$ f = 1
  flip x2 / v $\rightarrow$ f = 1
  flip x3 / v $\rightarrow$ f = 1
  $\rightarrow$ flip x1/b
  etc : ici ça marche avec b, v, r

Exercice : coloration

instanciation de départ 4 :j, 5 :r, 6 :j



Exercice : 4-reines

modèle avec une reine par colonne.
départ : x1=1, x2=2, x3=2, x4=4

- quel voisinage ?
- quelle fonction à évaluer ?
- déroulement

Exercice : voisinages multiples

- modèle simple : flip une variable
- mais ici on peut profiter de la contrainte “toutes les variables doivent être différentes” $\rightarrow$ comment ?
- résolution avec

1	2	3
4	5	6
7	8	9

Modélisation

"Team building" : 10 équipes de 6 employés.
30 nouveaux, 30 anciens, et les équipes doivent être 50/50.
Pas plus de 4 membres d'un même service dans une équipe
Pas de mélange entre les services A et B, ni E et F.
Certains nouveaux ont des "coachs" et doivent être dans la même équipe que leur coach.
L'employé 5 doit être mis avec 41 ou 51.
Numéros : pairs pour les nouveaux, impairs pour les anciens

Service	Numéros
A	0-19
B	20-39
C	40-44
D	45-49
E	50-54
F	55-59

Modélisation

quelques points à retenir de ce problème

- attention aux symétries dans les solutions possibles : facteur d'explosion combinatoire
- modélisations multiples : peut servir pour contraindre plus facilement les variables choisies $\rightarrow$ ajout de variables intermédiaires, et liens entre les points de vue
- autres exemples : n-reines, carré magique
- exemple de la contrainte all-different

Contrainte binaire / n-aire

passage de contrainte n-aire vers contrainte binaire : ajouter des variables

- à toute contrainte c d'arité > 2, associer une variable supplémentaire y,
- les domaines sont les tuples de variables concernées (x1,x2,x3,...)
- et pour toute variable x concernée, ajouter $x_i =$ argument_i(y)

Exemple / solution 1

4 variables binaires,

c1 : x1 ou x2 = 1

c2 : x1 et x2 = x3

c3 : x1 ou x2 = x4

ajout y1 pour c2, y2 pour c3

domaine y1 =

domaine y2 =

contraintes :

c2 remplacé par x1 = arg1(y1), x2 = arg2(y1), x3 = arg3(y1)

c3 remplacé par x1= arg1(y2), x2 = arg2(y2), x4 = arg3(y2)

Outils

- en tp, réimplémentation de certaines techniques
- mais des outils existent
  - langage de définition de contraintes (ex OPL, AMPL)
  - solveurs de contraintes (ex CPLEX)
- contraintes courantes implémentées (+ algo dédiés),
- ex de contraintes courantes : alldiff, séquence, etc

Modélisation : la répétition

Un concert est prévu avec 9 morceaux du durées différentes, impliquant différentes combinaisons parmi les 5 musiciens de l'orchestre. Les musiciens peuvent arriver aux répétitions juste avant le premier morceau qui les concerne et partir juste après le dernier.
Un problème d'optimisation serait de trouver un ordre de répétition des morceaux pour avoir le moins d'attente possible au total pour les musiciens, soit le temps pendant lequel un musicien est présent sans jouer (et est payé quand même).
Cf la table qui liste les musiciens par morceau, et la durée de chaque morceau.
Un niveau acceptable pour rester dans le budget est d'avoir un temps total de 20.

Modélisation : la répétition

Morceau	Mus. 1	Mus. 2	Mus. 3	Mus. 4	Mus. 5	Durée
1	1	1	1	1	0	2
2	1	1	1	0	0	4
3	0	0	0	0	1	1
4	1	1	0	0	0	3
5	0	1	0	1	1	3
6	1	1	0	1	1	2
7	1	0	1	0	1	5
8	0	1	1	0	1	7
9	1	0	0	1	0	6

Variables ?

ne pas hésiter à introduire des variables intermédiaires ...